

Bachelor Thesis

Automated on-screen eye movement analysis for the evaluation of medical device usability



Author

Carlos Augusto Guija Deza

Supervisor

Felix Wang

Professor

Mirko Meboldt

Bachelor Thesis

Automated on-screen eye movement analysis for the evaluation of medical device usability

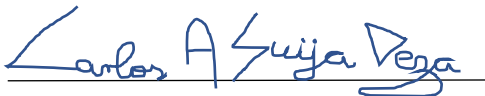
Dates

Start date 27.11.2020

End date 27.02.2020

Declaration of Originality

I hereby declare that I have written the present thesis independently and have not used othersources and aids than those stated in the bibliography.



Carlos Augusto Guija Deza

Confirmation

This thesis was written at and accepted by *pd|z* Product Development Group Zurich of ETHZurich.

Supervisor**Professor**

Preface

I would like to thank my supervisor Felix Wang for the patience and support he gave me throughout the thesis. He always gave me a helping hand. I also want to thank my mom for her moral support, for being there for me in the good and in the bad times and always pushing me to be better.

Carlos Augusto Guija Deza

Abstract

Usability Testing is indispensable in the development of medical devices. It validates intended functionality, guarantees patient safety, and provides a technological advantage. In this thesis, the performance of the automated Dynamic AOI Mapping algorithm (aDAM) is evaluated on two small medical devices. Its functionality is tested to see if it is suitable for its use in the growing field of eHealth applications. The performance showed very promising results. In the area of tracking, with the correspondent adjustments, a correct gaze point mapping can be achieved up to 98% of the cases. For Screen Matching purposes, a highly accurate matching can be acquired if the device plays a main role in the analyzed video. In case the images are from very low-resolution, the algorithm does not function correctly. Nevertheless, this algorithm performs well, and it can be used for the analysis of small medical devices.

Contents

1 Introduction	1
2 Materials and Methods	5
2.1 Devices.	5
2.2 Setup.	5
2.3 Algorithm	7
2.3.1 Preparation.	8
2.3.2 Raw Data Synchronization.	9
2.3.3 Detection and Tracking.	9
2.3.4 Gaze Point Mapping.	12
2.3.5 Screen Matching	12
2.3.6 Visualization	13
2.3.7 Mask R-CNN	14
2.4 Evaluations.	15
3 Results	16
3.1 First Time Use Problems	16
3.1.1 Required files.	16
3.1.2 Compilation Errors.	17
3.2 Tracking	18
3.2.1 Line Detection.	18
3.2.2 Ratios & Tolerances.	19
3.2.3 Occlusion.	20
3.2.4 Verification Bug.	21
3.2.5 Segmentation by coordinates.	22
3.3 Screen Matching.	26
3.3.1 High-Resolution Images.	26
3.3.2 Low-Resolution Images.	27
4 Discussion	29
4.1 Video scale adjustment.	29

4.2 Tracking.....	29
4.3 Screen Matching.....	29
4.4 Further steps.....	30
5 Conclusion	30
6 Bibliography	31

1 Introduction

Usability is defined by the Organization for Standardization as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” [1].

In the development of medical applications, Usability is a key component of good practice and it has been identified as an essential criterion for its assessment. [2]-[6]. A correct and intuitive design of the application is of major importance, since people who need it may have problems due to their health conditions [2].

By conducting Usability Evaluations, intended functionality is validated and patient safety is guaranteed, since it improves productivity, enhances user well-being, increases accessibility, and reduces the risk of harm (Maramba et al., 2019) [7].

In recent years, an exponential increase in the number of eHealth applications has taken place. Nevertheless, only a small fraction of the usability evaluation results has been published [7]. Usability Testing should be part of every design and development process, but it is often looked over due to the time and labor demands of implementing the same (Khasnis et al., 2019) [8].

Most of the published usability evaluation results relied on Traditional Usability Testing methods. The most used being of quantitative nature like questionnaires or task completion. Qualitative methods like think aloud protocol, focus groups and heuristic testing were also broadly used. The use of questionnaires does not necessarily pinpoint the problems that need to be addressed (Maramba et al., 2019). All these methods require specific input by the user and can have an evaluators bias making the process subjective and not transparent (Malan et al., 2018) [9]

Eye Tracking Technology allows to quantify cognitive processes and therefore provides valuable insights into the user’s thought processes during medical device usage. The collection and analysis of data do not require specific input by the user making it an objective procedure to evaluate the usability of a product (Mussnug et al., 2017)[10].

The evaluation of video material is a time-consuming activity (Kurzals et al., 2017) [11]. For this type of study, one has to go frame

by frame mapping the gaze point to an area of interest (AOI), which is considered tedious work. Costs are extremely high compared to the output and the whole process is therefore rarely used (Essig et al., 2010) [12].

Several methods aiming to automate the gaze object mapping, and therefore automating the usability evaluation have been proposed. The motivation behind this was to make eye tracking studies more accessible and practical as a method for usability testing (Malan et al., 2018). The goal is to prevent designers from avoiding usability evaluation due to the problems mentioned above (Khasnis et al., 2019).

In [9], a semi-automatic algorithm is proposed. In this algorithm, the evaluator has to define parameters (division of tasks, benchmark user to be followed) so that the program can analyze each subtask. Different measurements are compared, and the subtasks with largest deviation are displayed in the evaluators monitor for further inspection.

The authors (Shun Chiba et al., 2018) [13] also explained an algorithm that recognized the activity of the user with the help of the gazed text and viewpoint information. A fisheye camera was used for knowing the location of the user, and a mobile camera was used to identify the text with an optical character reader. The gathered information was then used to determine the activity and give assistance for the following steps.

In the presented thesis, the automated Dynamic AOI Mapping algorithm (aDAM) (Batliner et al., 2020) [14] will be evaluated in small medical mobile devices.

The advantage of this algorithm is that no static screen eye-tracking system needs to be used. It can track AOIs that change position over time (making them dynamic AOI's (Batliner et al., 2020)) by following the borders of the screen.

In addition, this algorithm can determine the content of the screen and is able to evaluate what part of the screen the user was looking at during device use.



Fig 1: (Left) Hamilton Ventilation system monitor and (right) Flowbox infusion device

In [14], the performance of the algorithm was evaluated on two medical devices shown in Fig 1. The algorithm successfully mapped the gaze point in over 97% of the cases and it identified the correct screen content using feature point matching in over 98% of the cases (Batliner et al., 2020).

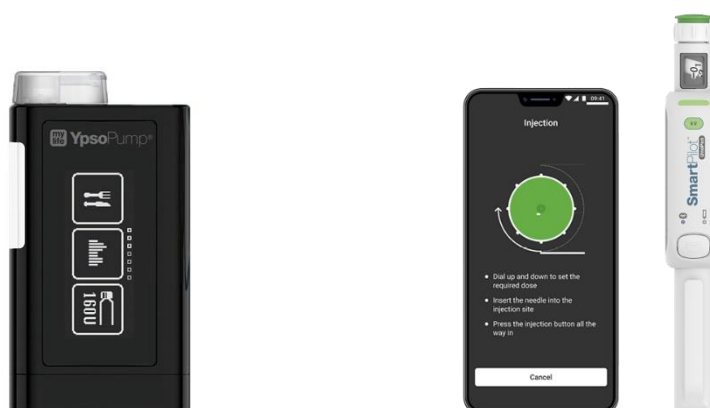


Fig 2: (Left) MyLife Ypsopump and (right) Ypsomed Delivery Systems Smartpilot

For this thesis, the performance of the algorithm was tested in smaller devices. These medical devices were the Ypsopump and the Smartpilot app for the Unopen syringe, both designed for the supply of insulin. The picture of each device is shown in Fig.2.

The motivation for this study was to test how well the performance of the algorithm would be in devices of reduced dimensions. In case of malfunctions, a respective solution would be provided. The goal was to get similar results as the already analyzed devices. That would be of significant value, since the field where the algorithm can be used for

usability testing would be very wide. As mentioned before, eHealth application is a growing field, and therefore of big interest as well.

In this work, the methodology, set up of the evaluations and an explanation of each program is provided. The results are shown and discussed before making a conclusion on the findings.

2 Materials and Methods

2.1 Devices

Fig.2 showed the two devices that were analyzed during the evaluation. The mylife Ypsopump is a small insulin pump designed for medical self-treatment. Its dimensions are (7.8 x 4.6 x 1.6 cm) and it has an OLED touchscreen that is used as menu.

The Ypsomed Delivery Systems Smartpilot is a complement for the variable dose pen for insulin UnoPen. It is used for a step by step patient guidance, tracks injections and provides log records information. It has a mobile application that runs in a Smartphone.

Both devices represent small cases in the spectrum of medical devices. Its evaluation is important since it shows how robust the algorithm really is. It also gives insights into possible applications for usability testing in the field of eHealth applications.

2.2 Setup

The laptop used to run the aDAM algorithm was an HP Omen 17 (2017) with a 2.5GHz Inter Core i5 processor. The code was run in an Anaconda Virtual Environment with python version 3.7.9. The main development tool was the OpenCV package (version 4.0.0) for computer vision tasks.

In order to use aDAM, the eye-tracking recording that will be analyzed is needed. This recording has to be in avi format. Other requirements are the layout image of the device and an image of every possible screen, all of them being in png format. The most crucial document is the raw data originated from the eye-tracking studies. The extracted data has to be presented in a csv format, where the coordinates of the gaze points at different times are shown. In another column, the type of Eye Movement is stated, where it could be either Fixation or Saccade.

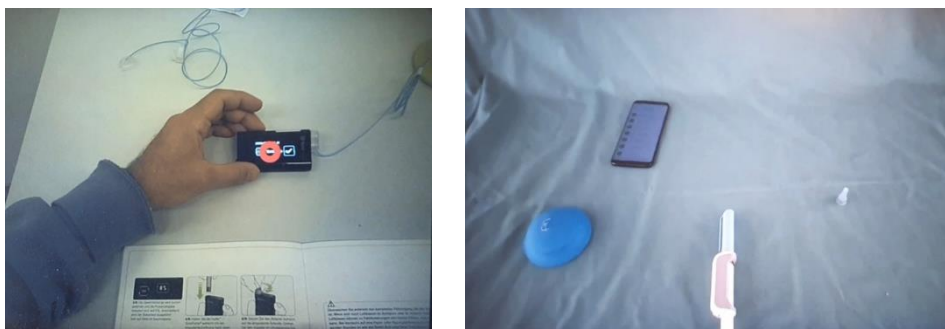


Figure 3: Extracted scene video of the (left) Ypsopump and (right) Smartpilot

Initially, all documents required for the evaluation of the Ypsopump were given. At a later point, the video and csv file for the Smartpilot were provided. The layout and screen images were then added. A frame of each video is shown in Fig. 3.

As mentioned before, there are two different functionalities that characterize aDAM. These are the tracking of the device, and the screen matching of its content. For a more in-depth analysis, new recordings were made, each of them to explicitly test the different parts of the algorithm.

Due to the pandemic, no actual experiment with eye tracking glasses was conducted. Instead, videos with the Samsung Galaxy s10e smartphone were recorded. For it to work, a dummy file was generated from the Ypsopump raw data. The entries had to be adjusted to the length of the video, and the gaze coordinates were fixed to a point that was on the device of interest.

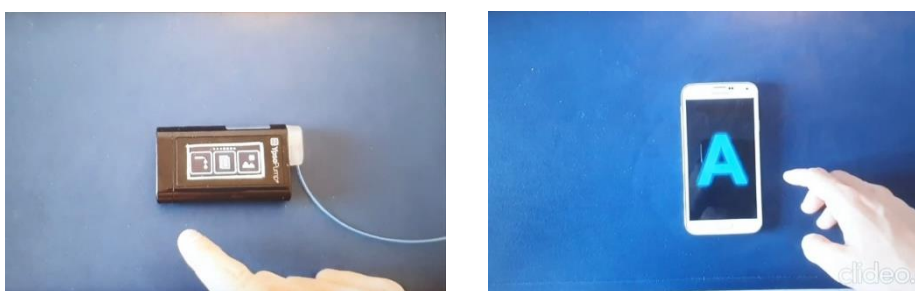


Figure 4: Extracted scene video of the (left) Ypsopump with tape and (right) Samsung Galaxy s5 with different screens.

These new recordings were of small duration making them easy to analyze. They also showed simple cases to test the different functionalities of the algorithm. In the case of the Ypsopump, tape was placed around the sides of the screen to see if an improvement of the tracking was possible. Videos with different types of screen were

recorded on the Samsung Galaxy s5 smartphone to test the screen matching process. A frame of each video is shown in Fig.4.

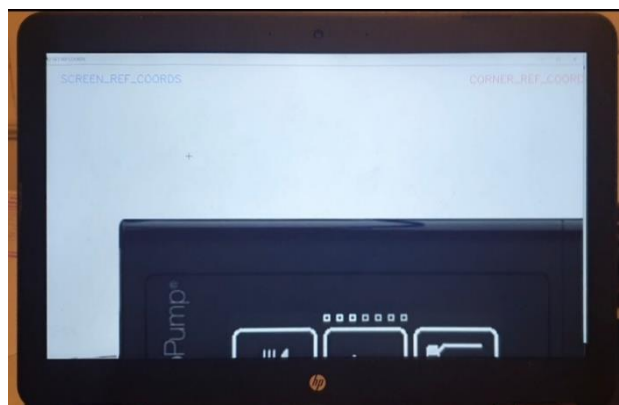


Figure 5: Half of the Ypsopump layout due to scale differences

During the evaluation it was noticed that the scale of video and images of the device were bigger than the laptop screen Fig.5. For that reason, a resizing of the materials was needed. When resizing the video, it is also necessary to adjust the gaze points in the csv file accordingly. In some cases, it was possible to conduct the study without the resizing of the video. It was later noticed that the scale of the video had an influence in the results obtained. The reasons for that are later explained. The important point is that the evaluation for each device was conducted twice. Once in the original version (1920 x 1080) if possible, and the other in the reduced version (960 x 540).

2.3 Algorithm

In this section the algorithm and key functions will be explained. It will be showed how the algorithm works when it is correctly executed. For a more detailed review please refer to Batliner et al. (2020) paper [14]. An overview is provided in Fig.6.

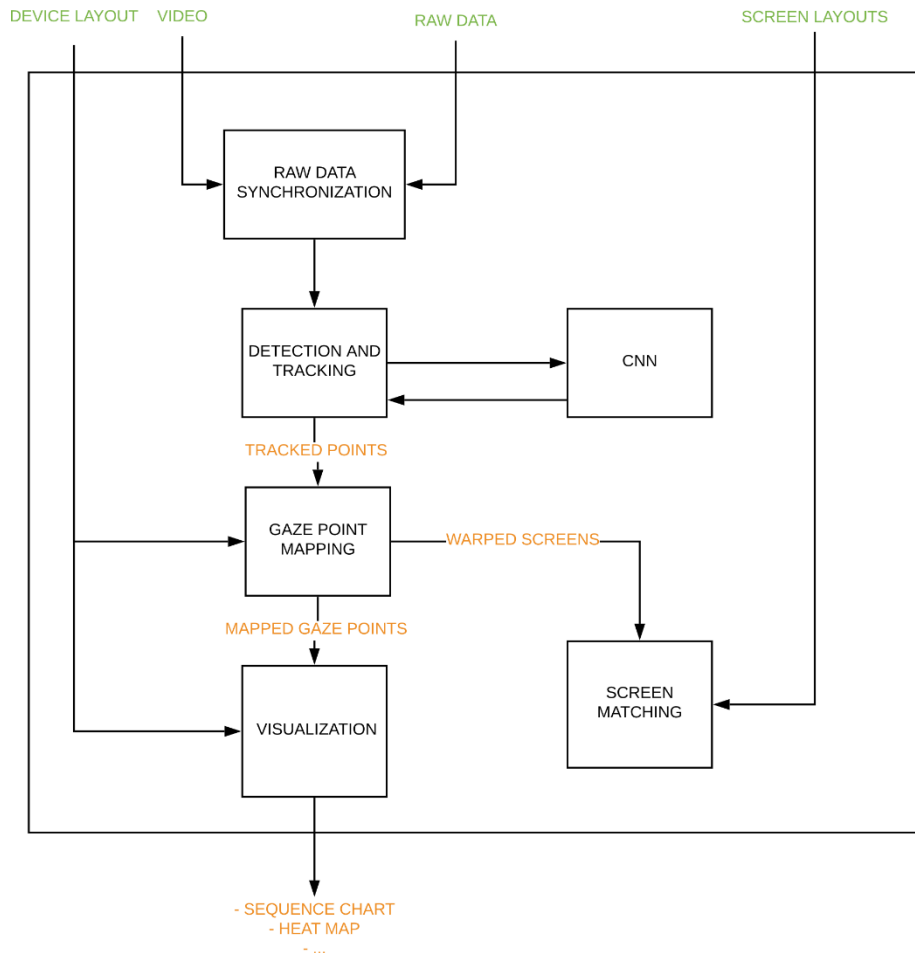


Figure 6: Overview of the algorithmic flow (Batliner et al., 2020)

2.3.1 Preparation

Before the execution of the different programs, it is checked that all required documents exist. The *Prep_win* program is responsible for that. A predefined structure of the different folders is required, and the program is in charge of placing the different files in the correspondent folder. This program is also responsible for defining the coordinates of the screen and the device in the layout image. These coordinates have to be given in clockwise direction. An image is provided in Fig. 7.



Figure 7: Screen (blue) and device (red) coordinates

2.3.2 Raw Data Synchronization

Before using the raw data extracted from the eye-tracking glasses, it is necessary to synchronize the obtained data with the recorded video. The *Raw_data_synchronization* program is responsible for that. The reason behind this, is that the video that has been recorded has a frame rate of 30fps, while the eye-tracking glasses have a sampling rate of 60Hz, resulting in more gaze points than frames in the video. The synchronization of both files results in a new csv file, where the different gaze points appear in the correspondent video frame.

2.3.3 Detection and Tracking

For the tracking part, the video and the synchronized file are needed. The *Tracking* program finds the coordinates of the device in each frame of the video. The mechanism used is the identification of its lines, which once identified, are used to calculate the intersections that represent the coordinates.

As initial information, one has to provide the ratio between the height and width of the device screen. It must also be specified whether a neural network is used or not. The first time the program is used, there is no neural network. This is created in the last step of the algorithm, and it is used to fully automate the tracking for next similar videos.

At the start of the program one can optionally choose the initial frame, which is sometimes necessary because the first frames could be black or not contain the device of interest. The first time the program is used, the process is semi-automatic.

In the first frame, one always has to give the coordinates of the device. Once this is done, the algorithm starts working. A first approximation of the coordinates in the next frame is made using the Lucas-Kanade algorithm [15]. This algorithm identifies the change of light intensity in the pixels to give its first approximation. This first approximation is not always accurate, but it gives an indication of where the coordinates could be. The first two steps are shown in Fig.8.



Figure 8: (left) Initial guess of device coordinates (right) Approximation of coordinates in the next frame using the Lucas-Kanade Algorithm.

The points of the first approximation are used to define a search area. An inner and outer tolerance are defined. A Canny edge detector (Canny (1987)) [16] is applied to the image. This detector identifies sharp discontinuities in a picture, giving as output the different curves in an image. The curves that are outside the tolerances are deleted, and only the lines of the device should remain. This process is showed in Fig.9.

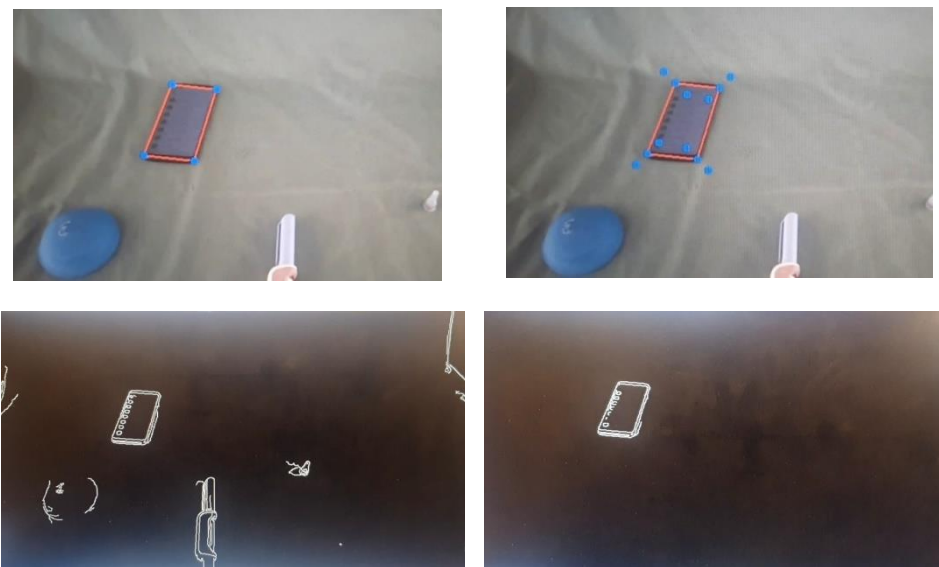


Figure 9: Upper row: (left) identification of search area and (right) definition of inner and outer tolerances. Lower row: Canny filter before (left) and after (right) the tolerances are applied

Once this is done, the smallest lines are filtered out, and the remaining lines are segmented twice. The first segmentation is given by the angle of inclination of the lines. The center of each line is calculated, and then a second segmentation with respect to the center points happen. The final result is four different groups of lines, each representing one side of the device. These steps can be seen in Fig 10.

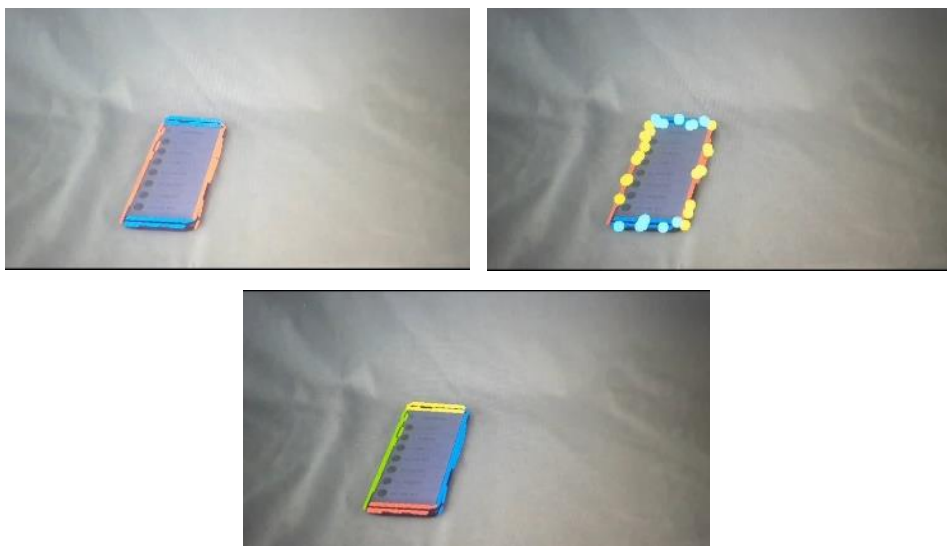


Figure 10: (upper left) Segmentation by angle, (Upper right) Center of lines, (Low row) Segmentation by coordinates.

The largest line of each group is selected and used to calculate the intersections. The intersections represent the coordinates of the device screen within the frame. It is checked that the found coordinates lie inside a ratio provided by the previous coordinates. If everything is in order, the found coordinates are used as input for repeating the process in the next frame. This continues until the end of the video. This is shown in Fig.11.

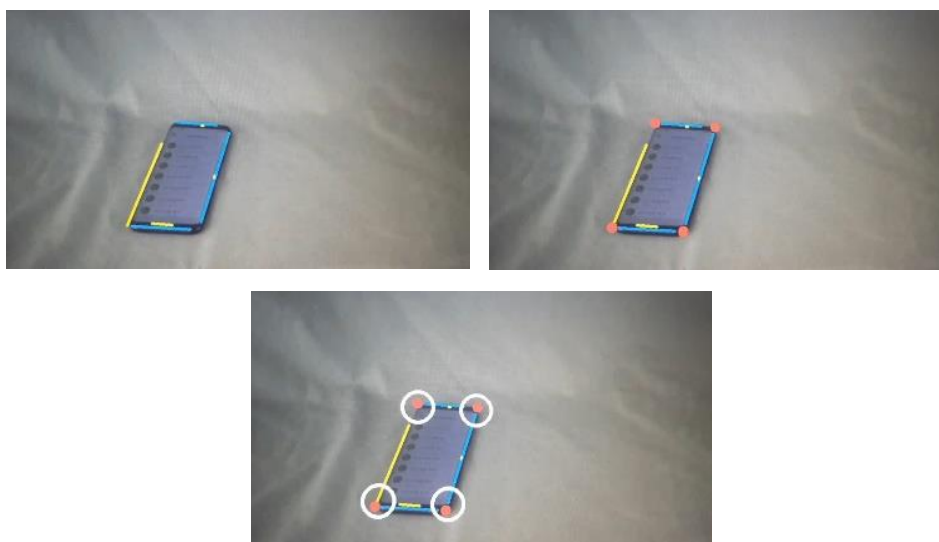


Figure 11: (upper left) Biggest line detection, (Upper right) Calculation of Intersections, (Low row) Verification of acceptable intersections

In case of malfunction, the algorithm stops, and the user has to provide the coordinates for the next frame manually. In case a neural network is used, Mask R-CNN is responsible for the initial guess of the coordinates.

At the end of the program, all the coordinates that were found are recorded and written to a csv file.

2.3.4 Gaze Point Mapping

With the last generated document, the next step is the extraction of the device screen content. This is done with the *Mapping* program. Using a transformation matrix, the detected screen and the gaze points are transformed to a frontal perspective, so that it can be displayed in the layout image. Only those frames where the gaze points are on the device screen will be saved for further process. In Fig. 12 the *Mapping* program can be seen.



Figure 12 (left) Mapped screen content without gaze point and (right) with gaze point

in order for the screen matching algorithm to be able to find the best match, when compare the detected screen with all possible device screens.

2.3.5 Screen Matching

During the screen matching step, the comparison of the gathered images is made. The idea is to compare the detected screen with all possible screen images and give as result the best match. The *screen_matching_parallel* program is run, where the software BRISK [17] is used. This software finds characteristic features in an image, which are usually either curves or corners. Those features are compared with the features of the other images, and the most similar result is returned and written in a new csv file. The comparison is seen in Fig. 13.



Figure 13 Characteristic feature detection and comparison

2.3.6 Visualization

After finishing the screen matching step, all the information that is required to illustrate the results in form of graphs and maps is at disposal.

First of all, the *create_aoi* program is used by the evaluator to choose the different AOI of the device. The *post-processing* program can be used to generate heat maps, focus maps, an image with the amount of time spent in each AOI and the sequence of the different AOIs viewed during the ET video. The *screen_post* program generates one more sequence, which shows the order of the viewed screens during the analysis. The different outputs are shown in Fig 14-16.



Figure 14: (left) Total amount of time spent in each AOI. (right) Location of the gaze points while watching the device. Results from Samsung Galaxy s5 analysis

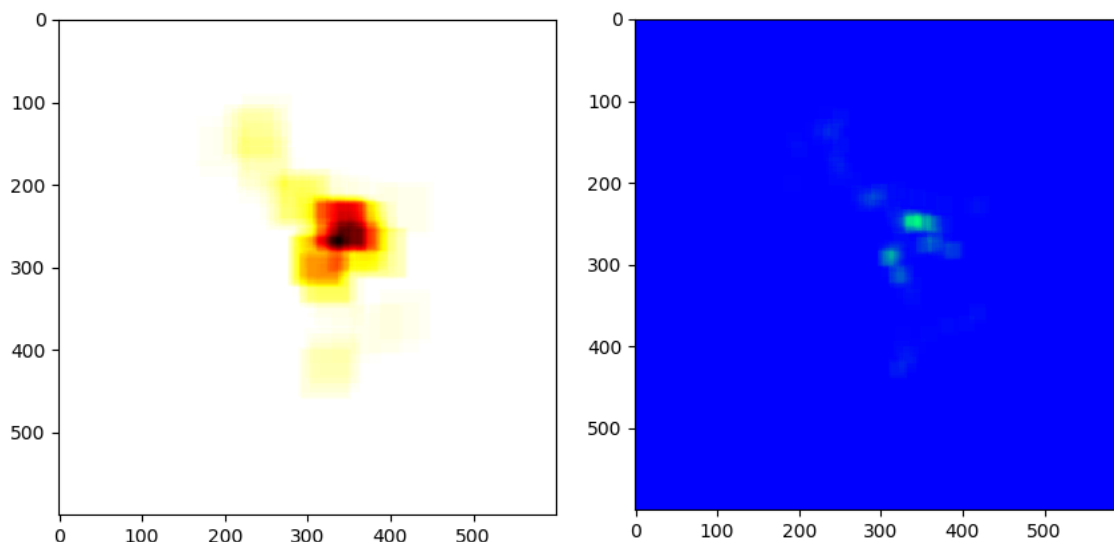


Figure 15: (left) Heat map and (right) Focus Map for dummy file video for Samsung Galaxy s5

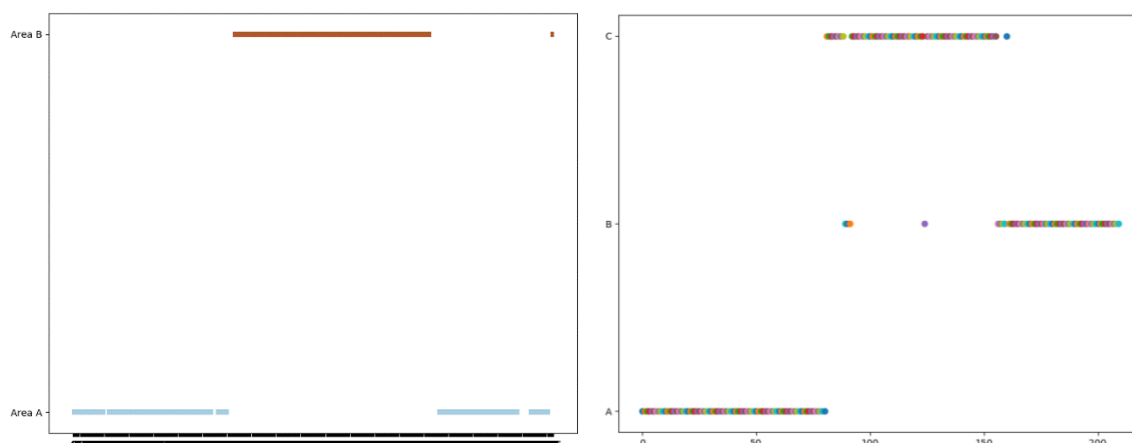


Figure 16: From Samsung Galaxy s5 results: (left) Sequence of viewed AOI (Blue: Area A, Red: Area B). (right) Sequence of viewed screens (Top: C, Middle :B, Low: A)

2.3.7 Mask R-CNN

Finally, one uses the *Device* program to train a neural network that will serve to make the process in similar videos fully automatic. This neural network is called Mask R-CNN. The *device* program has two functions, the first one is used to get a sample of coordinates to train the neural network, and the second function is the training of the neural network. The function of the neural network is to intervene whenever an error occurs and to give a first approximation of coordinates for the subsequent line identification. This is seen in Fig. 17.

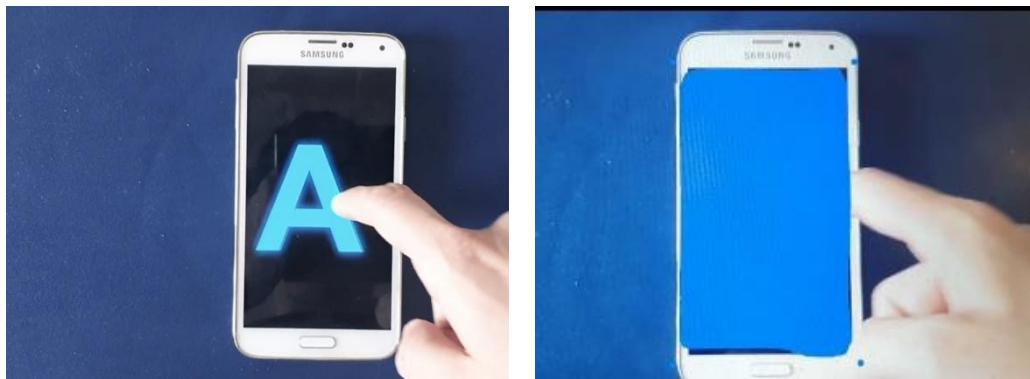


Fig 17: Device (left) before and (right) after the use of the Mask R-CNN in order to give the first approximation.

2.4 Evaluations

For this work a total of 5 different videos were analyzed. Three videos were with the Ypsopump: The original video, and two short videos focusing in the tracking and in the screen matching part. The other two videos were in a Smartphone. The first one using the Smartpilot, and the other one just looking at different screens. This distinction was made to assess functionality in simpler case scenarios before analyzing the original one, which was more time consuming.

All programs were run in all videos with the exception of the neural network creation. The most regular problems were noted during the evaluation and explained in the Results Section. Error rates regarding the tracking part for the different problems were made. Proposed solutions for each problem were given, and later discussed.

3 Results

In this section, the most relevant findings during the analysis of the medical devices are shown.

3.1 First Time Use Problems

3.1.1 Required files

As mentioned in Section 2.2, displaying the different images and videos was a problem due to its dimensions (Fig.5). Before the final solution, the code was changed to display the images and videos in a smaller format. The issue concerning that solution appeared during the training of the neural network, where the mask coordinates were proportionally smaller to the video. This can be seen in Fig.18.

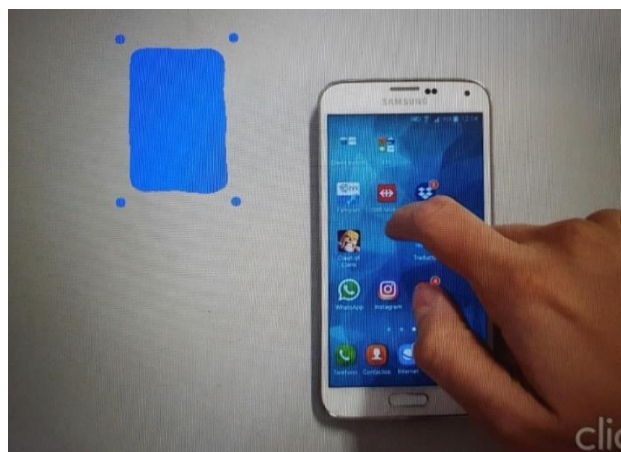


Figure 18: Error with Mask R-CNN

Due to this problem, the scale of all images, videos and documents were changed in a pre-processing step.

The first time the *raw_data_synchronization* program was executed for the Ypsopump, a problem due to the csv file happened. In the csv file, the gaze point information from multiple videos were included, and the algorithm was designed to only work with the gaze point information of the respective video. This caused a malfunction and to solve the problem a new csv file only with the information of one video was created.

3.1.2 Compilation Errors

The aDAM algorithm was originally designed for a Mac operating system (OS). For the conducted studies a Windows OS was used. This difference is relevant since the first time the programs were executed, different compilation problems happened. These problems can only be related to the change in OSs since in previous evaluations no such problems occurred. Solutions were provided and the problems didn't appear again during the evaluation.

In the *Tracking* program, a variable wasn't defined. In the *Mapping* program, an incorrect reading in the information provided by the csv file was made. While reading the coordinates of the gaze points, the title of each column was also taken into consideration, causing a malfunction since a string was tried to be read as an integer.

It is important to mention not to run the *Mapping* program multiple times as it doesn't overwrite the generated csv file, but adds new rows to it, causing a malfunction in the next steps.

For the *screen_matching_parallel* program an error happened with the *pool.map* function. This function wasn't able to recognize certain variables given as input. Although it couldn't recognize some variables, it could recognize an array, so the solution was to add these variables as extra elements of that array.

Finally, in the *device* program there was a bad delimitation of its different functionalities. This program has two functions. The first one is *Create*, in charge of collecting a sample of coordinates that will be used to train the neural network. The second function is *Train* with which the neural network starts training with the coordinates that are chosen. Those functions didn't have the proper delimitation, causing malfunctions when they were run. After improving the code structure, that problem didn't happen again.

After these changes, the evaluation of each device could begin. The most important problems occurred during the execution of the *tracking* and the *screen_matching_parallel* programs.

3.2 Tracking

3.2.1 Line Detection

During video analysis, the use of the Canny detector filter wasn't correctly applied at certain points. This incorrect behavior happened for different types of reasons.

The worst case was during the analysis of the Ypsopump. As intended, the screen coordinates were selected for the analysis. The difficulty was that the screen sides were not clearly visible. The fact that screen and device had the same color contributed for that problem. If no lines are detected, the algorithm stops working and waits for manual input. This problem can be seen in Fig.19.



Figure 19: (left) screen coordinates for Ypsopump evaluation and (right) detected lines after the Canny Filter was used

Since it was intended to select the screen coordinates and not the device coordinates, a new video was generated. This was made to avoid occlusion as much as possible, and tape was used to show the different sides of the screen. The use of tape made the Canny detector work as intended. This can be seen in Fig. 20.



Figure 20: (left) evaluation of Ypsopump with tape and (right) detected lines with Canny Detector Filter

For the Analysis of the Samsung Galaxy s5 no problem was noticed, as its design clearly differentiates screen from device. This is seen in Fig. 21.

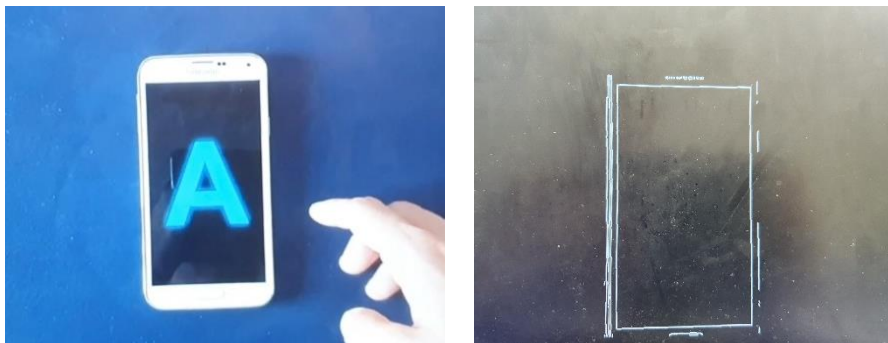


Figure 21: (left) evaluation of Samsung Galaxy s5 and (right) detected lines with Canny Detector filter

During the Smartpilot video analysis, a light source was responsible for the shadows on one side of the device. In those situations, the Canny Filter wasn't able to detect the corresponding side. The only solution was to fast-forward the video to a point where that shadow didn't exist. In the reduced format of the video that problem didn't happen so often. The reason was that the shadow was interpreted as part of the line. This can be seen in Fig.22.



Figure 22: (left) evaluation of Smartpilot app and (right) detected lines with Canny Detector filter

3.2.2 Ratios & Tolerances

For tracking the device, the ratio between height and width is given as input. In order to continue the analysis in the next frame, it is always checked that the ratio doesn't deviate much from the given input. If the ratio is very different, the algorithm stops working and manual intervention is required.

When the device is not seen from a frontal perspective, some sides can look bigger than others and therefore cause a malfunction.

In the same way, when the segmentation by angle is made, it is always checked that the angle between the vertical and horizontal lines is at least form 70 degrees. If that is not the case, nothing is returned, and a

manual intervention is needed as well. This happened frequently in devices seen from non-frontal perspectives.

The solution for this problem was to change the tolerances from these checkpoints. For the ratio, the tolerance was increased from 20% to 50%. For the angle tolerance, the angle difference changed from 70 degrees to 25 degrees. After these changes, these problems didn't happen again.

3.2.3 Occlusion

One of the problems where no solution was found was during the occlusion of the device. The algorithm is robust enough to correctly detect the lines in the presence of small obstructions. This can be seen in Fig. 23.

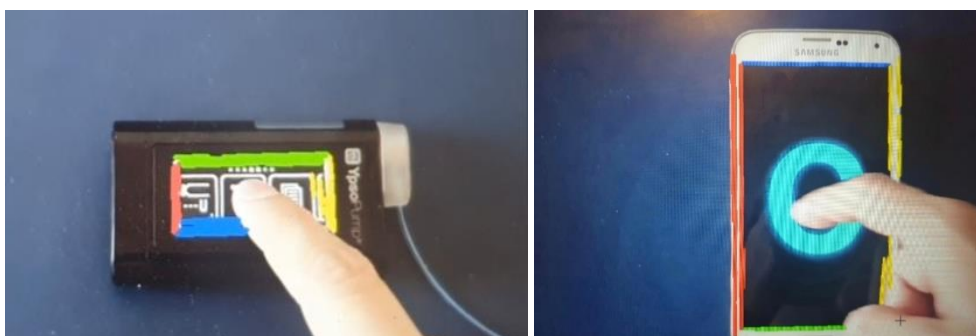


Figure 23: Correct line identification and segmentation in (left) Ysopump and in (right) Samsung Galaxy s5 with the presence of a finger

The problem happens when the finger or the palm of the hand completely covers one of the sides of the device. This can also happen when due to head movements, part of the device is left out of the image. When this situation happens, the line identification and segmentation is completely wrong. Either not all sides are detected, or some small lines that do not correspond to the side of the device are used and give a wrong intersection. This can be seen in Fig.24.



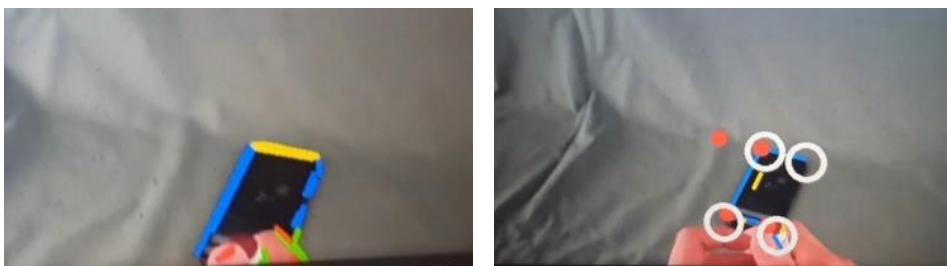


Figure 24: Upper row: Complete occlusion of right side of the Ypsopump resulting in a wrong segmentation. Lower row: Complete occlusion of bottom side of the Smartpilot App, causing a (left) wrong segmentation and (right) wrong intersection.

When this type of error happened, the only solution was to fast-forward the video until all four sides were clearly visible again.

3.2.4 Verification Bug

During the analysis of the Smartpilot a problem in the *is_ok_location* checkpoint happened. This checkpoint verified that the founded coordinates were in a certain distance with respect to the old ones.

An error message appeared in the console, although the coordinates were inside the ratio, which caused the algorithm to stop, and made the process manual. This can be seen in Fig. 25.

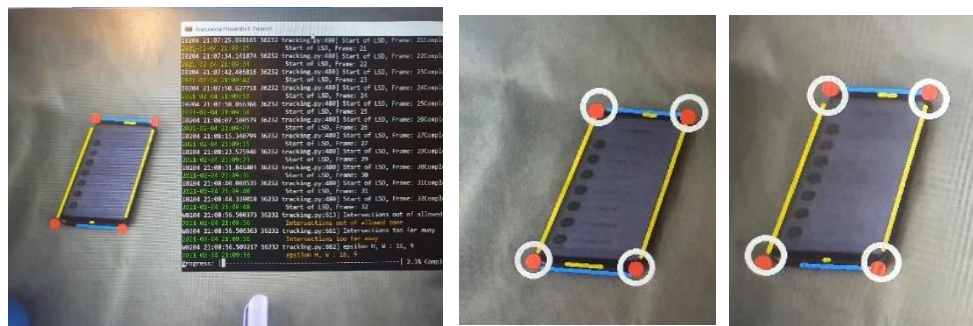


Figure 25: (Left) *is_ok_location* error message in the console and (middle, right) the respective verifications that showed that the points were inside the tolerance (white circle).

It was later discovered that the problem didn't lie in the tolerance of the ratio. The problem lied in the sorting of the points, that swapped the positions of the bottom coordinates, and making them seem as they had a huge movement.

For that discovery, the checkpoint was briefly removed, and it was seen that in later frames the search area was in form of an hourglass,

preventing the program of finding the vertical lines and thus stopping the algorithm again. This can be seen in Fig.26.



Figure 26: (Upper row) Search area identification after ignoring the `is_ok_location` checkpoint. (Lower row) Line detection with Canny filter with the respective search area.

The code structure of the sorting mechanism was changed, and the problem didn't happen again.

3.2.5 Segmentation by coordinates

The most frequent problem occurred with the second segmentation. In the segmentation by coordinates, the center points of the vertical and horizontal lines are used as input to calculate two different groups. Respectively left and right, and then top and bottom line. The clustering algorithm used for this task is the K-Means method [18].

When using the K-Means method there are two valid outcomes. The first outcome is the correct segmentation that gives the different lines.

The second outcome is the incorrect segmentation, where in one side of the device, two different groups of lines are detected. This type of problem tends to happen independently of occlusions as seen in Fig. 27.

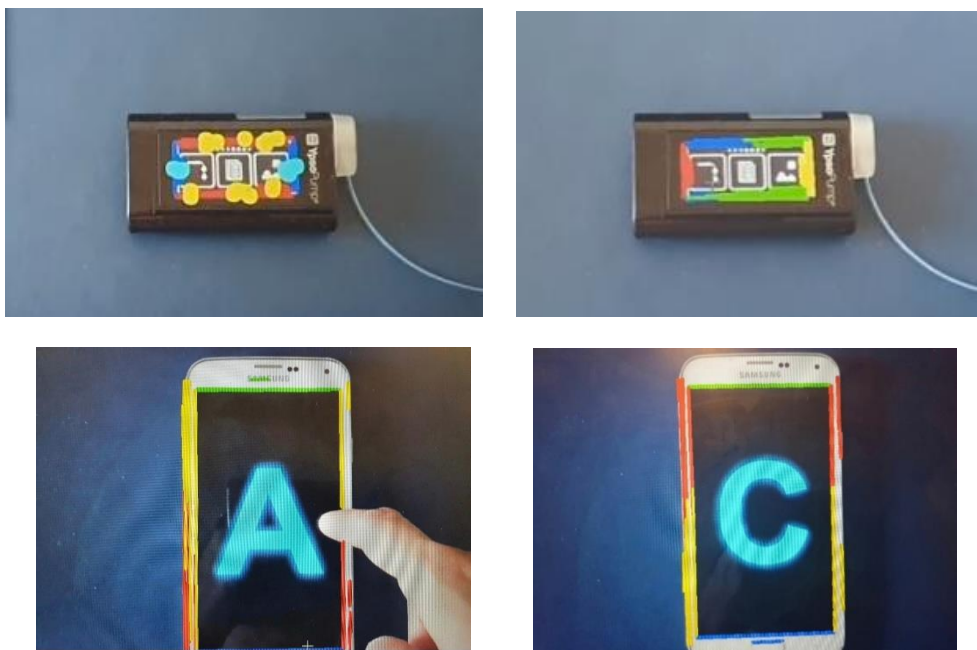


Figure 27: (Upper row) Identification of center points in Ypsopump and the correspondent incorrect segmentation by coordinates. (Lower row) Incorrect segmentation by coordinates with finger (left), and without finger (right).

Although this problem happens frequently, the error message doesn't appear so often, since in many occasions the chosen lines for each group are in opposite sides.

This was a very important problem for the Ypsopump since it made the process almost manual. It was noticed that the reduction of the video dimensions improved the tracking of the device, and it was enough for a correct execution of the Smartpilot video.

Unfortunately, that wasn't enough for the Ypsopump. At the beginning, different types of clustering algorithms were implemented to see if an improvement was reached [19]-[27]. As seen in Table 1, each algorithm was tested in the first 25 frames of the Ypsopump. The algorithms without number weren't suited for clustering, as malfunctions occurred during the whole process. Although there are some algorithms that reported only 5 interventions, most of the times the segmentation was still incorrect. Better results were only achieved since the biggest lines were in opposite sides. The most promising algorithm was Mean-Shift. This algorithm did not only show better results, but the segmentation was correctly done. Unfortunately, it was later seen that its implementation in the other devices showed worse results than the K-Means, and therefore wasn't included.

	Interventions
<i>K-Means</i>	8
<i>BIRCH</i>	4
<i>Affinity Propagation</i>	-
<i>Agglomerative Clustering</i>	5
<i>DBSCAN</i>	-
<i>MiniBatch K-Means</i>	5
<i>Mean Shift</i>	3
<i>OPTICS</i>	-
<i>Spectral Clustering</i>	-
<i>Gaussian Mixture Model</i>	5

Table 1: Number of interventions due to bad segmentation in the first 25 frames for the Ypsopump evaluation

To solve the problem, fewer center points were required for a better segmentation. To achieve this, a filtering of bigger lines was made. In previous evaluations, lines whose length was up to 6% of the total length were erased. Thanks to the tape, a bigger filtering could be achieved, and it was set up to 30% of the total length.

It is important to mention, that in other devices this may have not worked out properly, since bigger filtering could erase important information, like the lines on one side. It is important to choose the correct filtering for each device. After the filtering, the tracking program worked correctly.

After considering all required changes for a correct tracking, the evaluations were made. In the following Tables, the best results achieved for each video are shown. The different errors, its amount and total error rate are shown. A feature of the algorithm is that if an error is constantly repeated, the video is fast-forwarded until the moment when the two previous images have no gaze points. That's the reason why in some videos the amount of analyzed frames is different.

	Smartpilot (1920 x 1080)	Smartpilot (960 x 540)
<i>Total number of frames</i>	1452	
<i>Total amount of analyzed frames</i>	457	1452
<i>Line Detection</i>	7	2
<i>Occlusion</i>	3	10
<i>Height and width Ratio</i>	0	3
<i>Incorrect second segmentation</i>	5	11
<i>Out of allowed zone</i>	2	0
<i>Total Error Rate</i>	3.71%	1.79%

Table 2: Total amount of analyzed frames, amount of errors and error rate for the evaluation of the Smartpilot in original and reduced size.

	Samsung galaxy s5 (1920 x 1080)	Samsung galaxy s5 (960 x 540)
<i>Total number of frames</i>	437	
<i>Total amount of analyzed frames</i>	374	437
<i>Line Detection</i>	2	0
<i>Occlusion</i>	1	3
<i>Height and width Ratio</i>	0	0
<i>Incorrect second segmentation</i>	26	7
<i>Out of allowed zone</i>	3	5
<i>Total Error Rate</i>	8.28%	3.43%

Table 3: Total amount of analyzed frames, amount of errors and total error rate for the evaluation of the Samsung Galaxy s5 in original and reduced size.

	Ypsopump Tracking (960 x 540)	Ypsopump Screen Matching (960 x 540)
<i>Total number of frames</i>	533	387
<i>Total amount of analyzed frames</i>	533	320
<i>Line Detection</i>	0	0
<i>Occlusion</i>	6	9
<i>Height and width Ratio</i>	0	0
<i>Incorrect second segmentation</i>	26	33
<i>Out of allowed zone</i>	0	0
<i>Total Error Rate</i>	6%	13.125%

Table 4: Total amount of analyzed frames, amount of errors and total error rate for the evaluation of the Ypsopump with tape in reduced size version for Tracking and for Screen Matching

3.3 Screen Matching

For the Screen Matching part, a very important malfunction occurred with low- resolution images. During the execution of this program, a distinction between the different analyzed videos had to be made. Results varied a lot depending on the resolution of the extracted images and they had to be properly addressed.

3.3.1 High-Resolution Images

The videos recorded to test simple cases, showed very high-quality images since the device was always centered and conformed an important part of the video. High-resolution images resulted in highly accurate matches.

In the case of the Samsung Galaxy s5, better results were obtained in the reduced size video. The main difference was noticed during the transition between images, where the original version had more mismatches than the reduced version. The comparison is seen in Fig.28.

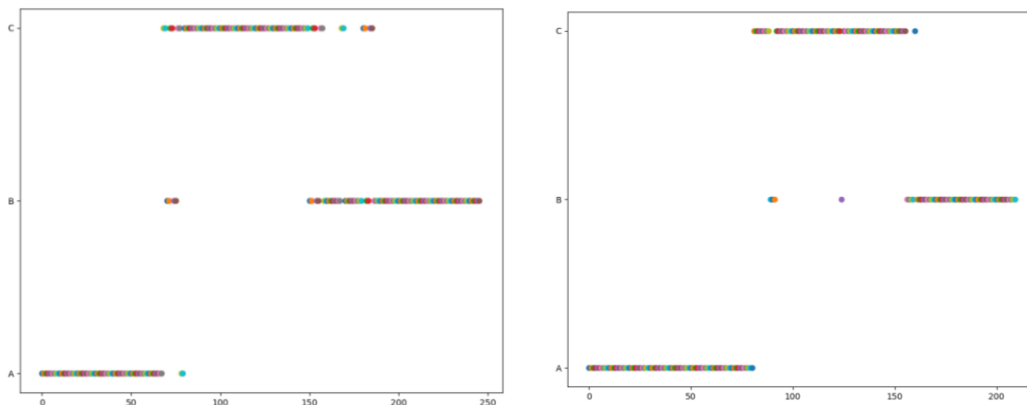


Fig 28: (left) screen sequence from original video (right) screen sequence from reduced size video.

3.3.2 Low Resolution Images

For the Smartpilot, the device conformed only a small part of the video. The device was in a secondary plane and was seen in a non-frontal perspective. The ratio of device size to video size plays a very important role and can result in either very high- or low-resolution images. In the case of the Smartpilot, low resolution images were obtained, which lead to a malfunction.

To understand this malfunction, one has to understand how the process of comparing images works.

The BRISK software [17] detects characteristic features in those images that are analyzed. These can be corners or curves, and the results are stored in an array called “*Descriptor*”. When comparing images, their “*Descriptors*” are compared, and from that comparison the similarities are extracted. The result with the most matches is returned as the solution.

However, the quality of certain images was so low, that the software was not able to find sufficient features to add to the *Descriptor*. During the analysis of the Smartpilot it was discovered that in 8 of the 376 extracted images that was the case. This can be seen in Fig. 29. When this happened, the descriptor was not an empty array, but was classified as a NoneType variable, which led to the malfunction.

This problem was easily solved by changing the structure of the code.

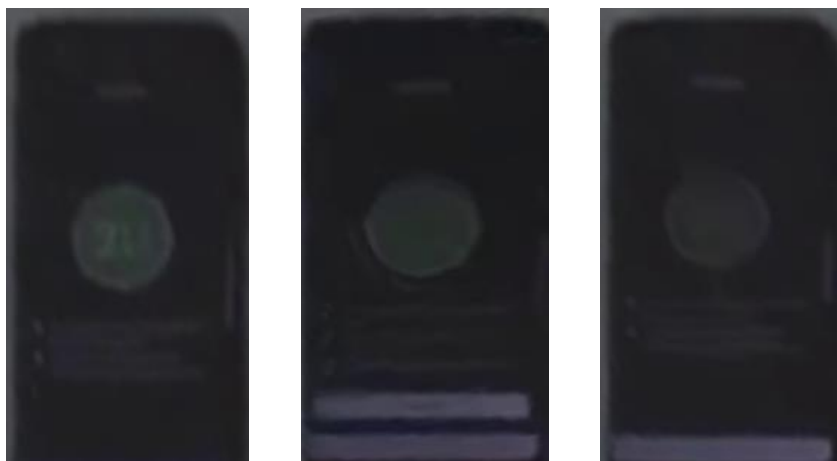


Fig 29: Low resolution images classified as NoneType.

The resolution of the images was very low, and therefore the Screen Matching was incorrect. In Fig. 30 it is seen the correct sequence of screens that should be obtained with the obtained sequence.

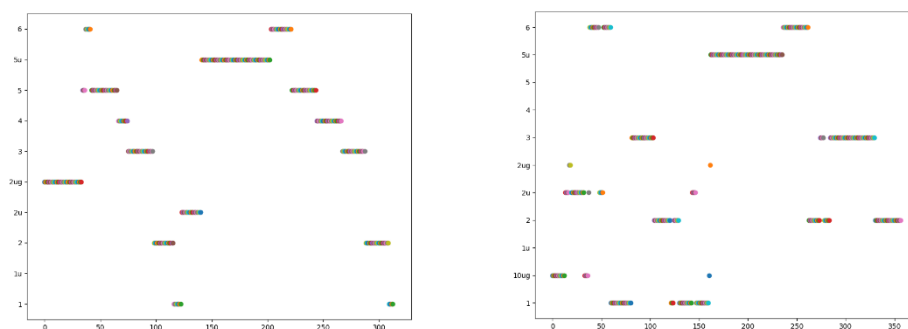


Fig 30: (left) Correct screen sequence of Smartpilot video. (right) Screen sequence given as output for the Smartpilot video.

This posed a serious problem for the purpose of the algorithm. The only solution that was found was the substitution of the screen folder images with one equivalent coming from the extracted images about to be analyzed.

After this change, the screen matching was perfect, but using the resource of manually changing the images in the folder goes against the idea of an automated process.

4 Discussion

4.1 Video scale adjustment

During the different evaluations it was explained that the analyzed video had to be reduced in size due to display reasons. In the Results section it was seen that the small video format achieved better results. A first factor was that in the reduced version, small shadows were not a problem while applying the Canny filter and thus being able to identify the lines properly. That is something that played a huge roll while analyzing the Smartpilot video, preventing it from fast-forwarding to the end. Another reason was that in the small version of the video, less lines were found, which meant less center point and therefore a better segmentation by coordinates. This is the reason why the tracking of the device had less error messages.

A downside of making the video smaller would be the compression of the extracted images, which could lead to worse screen-matchings. Of course, if the device is centered in the video, the screen matching part is even better than the original, since less feature points are considered, and outliers are from less influence.

4.2 Tracking

There were different kind of problems during the Tracking part of the algorithm, but the results showed that the error rate was very small and therefore suited for the tracking of the small medical devices.

It has to be considered that for properly execution, some adjustments have to be made, either by the reduction in the dimensions of the video, the use of tape around the sides for better line identification, the change in tolerances or the length of the filtered lines. There are many parameters with which the results can be improved. The only downside is that the filtering of lines is a trial and error measurement, which could consume useful time.

4.3 Screen Matching

As mentioned in the Results section, with high resolution extracted images, a highly accurate screen matching can be obtained. This will always be the case when the device plays an active role in the analyzed video. If the device is in a second plane, if it is only seen in a small fraction of the size video, low-resolution images will be obtained, and an incorrect screen matching will be made.

The founded solution was to manually intervene in the folder organization and provide extracted images as screen images for a better matching. Although after the interference the matching is highly accurate, the intervention diminishes the purpose of an automated usability algorithm. Instead of the founded solution, an improvement on the screen matching program could be made, so that this kind of problem doesn't tend to happen.

4.4 Further steps

This algorithm has a lot of potential and is suitable for small medical devices in normal circumstances. The robustness in the algorithm could increase in case an alternative method for the ratio inspection is found. In that way, in case of occlusion the visualized part of the device would be sufficient.

Another initiative would be an improvement of the screen-matching program, so that low-resolution images could be better matched, or that at least similar screens can still be differentiated.

The different maps, charts, and sequences that the algorithm gives as output can be used for usability evaluation. On the other hand, graph sequences are barely used and get more complex the more screens there are, or the larger the video gets. What could be done is use the output from the aDAM algorithm as input for another algorithm for a better understanding of the results.

5 Conclusion

In this thesis the performance of the automated Dynamic AOI Mapping algorithm (aDAM) was evaluated in two different medical devices. A total of 6 different videos were evaluated. These medical devices were from small dimensions, and the functionality of the algorithm was tested. The motivation was the growing field in eHealth applications, where the algorithm could potentially be used for usability testing reasons. The results showed that with the proper adjustments a very accurate tracking of the device can be made. Results showed an accuracy up to 98%. For simple cases where the device plays an active role in the analyzed video, the screen matching can be very accurate. Videos where the device is left in a second plane do not perform well during the screen matching, since low-resolution images are extracted. Only with a manual intervention high results can be obtained as well, but that goes against the idea of an automated process. The algorithm performs well in both devices, but for it to work properly, some

adjustments like video dimensions, the use of tape in the sides or line filtering are very useful. Further improvements can be made in the algorithm for more robustness in the areas of screen matching or clustering. With all things taken into consideration, this is an approach that can play a very important role for eye tracking usability testing.

6 Bibliography

- [1] A. Dix, Human-computer interaction, *Encycl. Database Syst*, Springer, 2009, pp.1327–1331
- [2] B.C. Zapata, J.L. Fernandez-Aleman, A. Idri, A. Toval, Empirical studies on usability of mHealth apps: a systematic literature review, *J. Med. Syst.* 39 (2015) 1, URL: <https://doi.org/10.1007/s10916-014-0182-2>.
- [3] N.H.S. England, Health Developer Network Digital Assessment Questions - Beta -Health Developer Network (n.d.). (2019) (Accessed July 5, 2018) URL: <https://developer.nhs.uk/digital-tools/daq/>.
- [4] MHRA, Human Factors and Usability Engineering - Guidance for Medical Devices Including Drug-device Combination Products, *Med. Healthc. Prod. Regul. Agency*, 2017, pp. 1–30 URL: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/645862/HumanFactors_Medical-Devices_v1.0.pdf.
- [5] ORCHA, About, (2019) (n.d.). URL: <https://www.orchac.co.uk/about/>(Accessed July 5, 2018).
- [6] Our Mobile Health, App Library, www.ourmobilehealth.com, (.d.). (2019)(Accessed July 6, 2018), URL: <https://www.ourmobilehealth.com/app-library.html>.
- [7] MARAMBA, Inocencio; CHATTERJEE, Arunangsu; NEWMAN, Craig. Methods of usability testing in the development of eHealth applications: a scoping review. *International journal of medical informatics*, 2019, vol. 126, p. 95-104.
- [8] KHASNIS, Shubhangi S., et al. Analysis of automation in the field of Usability Evaluation. En 2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE). IEEE, 2019. p. 85-91

- [9] MALAN, Katherine M.; ELOFF, Jan HP; DE BRUIN, Jhani A. Semi-automated usability analysis through eye tracking. *South African Computer Journal*, 2018, vol. 30, no 1, p. 66-84.
- [10] MUSSGNUM, Moritz, et al. *Mobile eye tracking in usability testing: designers analysing the user-product interaction*. 2015.
- [11] Kurzhals, K, Hlawatsch, M, Seeger, C, et al. (2017) Visual analytics for mobile eye tracking. *IEEE Transactions on Visualization and Computer Graphics* 23, 301–310. [CrossRef](#) [Google Scholar](#) [PubMed](#)
- [12] Essig, K, Sand, N, Schack, T, et al. (2010) Fully-automatic annotation of scene videos: establish eye tracking effectively in various industrial applications. *SICE Annual Conference 2010*. Piscataway, New Jersey, US: IEEE, pp. 3304–3307. [Google Scholar](#)
- [13] 10. CHIBA, Shun, et al. Activity recognition using gazed text and viewpoint information for user support systems. *Journal of Sensor and Actuator Networks*, 2018, vol. 7, no 3, p. 31.
- [14] BATLINER, Martin, et al. Automated areas of interest analysis for usability studies of tangible screen-based user interfaces using mobile eye tracking. *AI EDAM*, 2020, vol. 34, no 4, p. 505-514.
- [15] Lucas, B. D. & Kanade, T. (1981), 'An Iterative Image Registration Technique with an Application to Stereo Vision.', *Robotics* 81, 674--679. URL: <http://cseweb.ucsd.edu/classes/sp02/cse252/lucaskanade81.pdf>
- [16] Canny, J. (1987), 'A Computational Approach to Edge Detection', *Readings in Computer Vision* pp. 184--203. URL: <https://www.sciencedirect.com/science/article/pii/B9780080515816500246>
- [17] LEUTENEGGER, Stefan; CHLI, Margarita; SIEGWART, Roland Y. BRISK: Binary robust invariant scalable keypoints. En 2011 *International conference on computer vision*. Ieee, 2011. p. 2548-2555.
- [18] Jain, A. K. (2010), 'Data clustering: 50 years beyond K-means', *Pattern Recognition Letters* 31(8), 651--666. URL: <https://www.sciencedirect.com/science/article/pii/S0167865509002323>
- [19] ZHANG, Tian; RAMAKRISHNAN, Raghu; LIVNY, Miron. BIRCH: an efficient data clustering method for very large databases. *ACM sigmod record*, 1996, vol. 25, no 2, p. 103-114.

- [20] FREY, Brendan J.; DUECK, Delbert. Clustering by passing messages between data points. *science*, 2007, vol. 315, no 5814, p. 972-976.
- [21] MURTAGH, Fionn; LEGENDRE, Pierre. Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion?. *Journal of classification*, 2014, vol. 31, no 3, p. 274-295.
- [22] ESTER, Martin, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *En Kdd*. 1996. p. 226-231.
- [23] SCULLEY, David. Web-scale k-means clustering. *En Proceedings of the 19th international conference on World wide web*. 2010. p. 1177-1178.
- [24] COMANICIU, Dorin; MEER, Peter. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 2002, vol. 24, no 5, p. 603-619.
- [25] ANKERST, Mihael, et al. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record*, 1999, vol. 28, no 2, p. 49-60.
- [26] NG, Andrew Y., et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2002, vol. 2, p. 849-856.
- [27] REYNOLDS, Douglas A. Gaussian Mixture Models. *Encyclopedia of biometrics*, 2009, vol. 741, p. 659-663.